In [5]:

```
from skimage.color import rgb2gray
from skimage import data
import matplotlib.pyplot as plt
import numpy as np
from scipy.linalg import svd
X = np.array([[1,1],[0,1],[-1,1]])
print(X)
print("")
print("")
print("")
print("")
U, singular, V_transpose = svd(X)
# print different components
print("U: ", U)
print("")
print("")
print("")
print("")
print("Singular array", singular)
print("")
print("")
print("")
print("")
print("V^{T}", V_transpose)


"""
Calculate Pseudo inverse
"""
# inverse of singular matrix is just the reciprocal of each element
singular_inv = 1.0 / singular
# create m x n matrix of zeroes and put singular values in it
s_inv = np.zeros(X.shape)
s_inv[0][0] = singular_inv[0]
s_inv[1][1] = singular_inv[1]
# calculate pseudoinverse
M = np.dot(np.dot(V_transpose.T, s_inv.T), U.T)
print(M)
```

```
[[ 1  1]
 [ 0  1]
 [-1  1]]
```

```
U:  [[-0.57735027 -0.70710678  0.40824829]
 [-0.57735027  0.         -0.81649658]
 [-0.57735027  0.70710678  0.40824829]]
```

```
Singular array [1.73205081 1.41421356]
```

```
V^{T} [[-0. -1.]
 [-1. -0.]]
[[ 0.5         0.         -0.5       ]
 [ 0.33333333  0.33333333  0.33333333]]
```

In [6]:

```
from skimage.color import rgb2gray
from skimage import data
import matplotlib.pyplot as plt
import numpy as np
from scipy.linalg import svd
X = np.array([[3,2,2],[2,3,-2]])
print(X)
print("")
print("")
print("")
print("")
U, singular, V_transpose = svd(X)
# print different components
print("U: ", U)
print("")
print("")
print("")
print("")
print("Singular array", singular)
print("")
print("")
print("")
print("")
print("V^{T}", V_transpose)


"""
Calculate Pseudo inverse
"""
# inverse of singular matrix is just the reciprocal of each element
singular_inv = 1.0 / singular
# create m x n matrix of zeroes and put singular values in it
s_inv = np.zeros(X.shape)
s_inv[0][0] = singular_inv[0]
s_inv[1][1] = singular_inv[1]
# calculate pseudoinverse
M = np.dot(np.dot(V_transpose.T, s_inv.T), U.T)
print(M)
```

```
[[ 3  2  2]
 [ 2  3 -2]]
```

```
U:  [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

```
Singular array [5. 3.]
```

```
V^{T} [[ 7.07106781e-01  7.07106781e-01  3.67439059e-16]
 [-2.35702260e-01  2.35702260e-01 -9.42809042e-01]
 [-6.66666667e-01  6.66666667e-01  3.33333333e-01]]
[[ 0.15555556  0.04444444]
 [ 0.04444444  0.15555556]
 [ 0.22222222 -0.22222222]]
```

In [10]:

```python
# Python3 Program to decompose
# a matrix into lower and
# upper triangular matrix
MAX = 100


def luDecomposition(mat, n):

    lower = [[0 for x in range(n)]
             for y in range(n)]
    upper = [[0 for x in range(n)]
             for y in range(n)]

    # Decomposing matrix into Upper
    # and Lower triangular matrix
    for i in range(n):

        # Upper Triangular
        for k in range(i, n):

            # Summation of L(i, j) * U(j, k)
            sum = 0
            for j in range(i):
                sum += (lower[i][j] * upper[j][k])

            # Evaluating U(i, k)
            upper[i][k] = mat[i][k] - sum

        # Lower Triangular
        for k in range(i, n):
            if (i == k):
                lower[i][i] = 1 # Diagonal as 1
            else:

                # Summation of L(k, j) * U(j, i)
                sum = 0
                for j in range(i):
                    sum += (lower[k][j] * upper[j][i])

                # Evaluating L(k, i)
                lower[k][i] = int((mat[k][i] - sum) /
                                  upper[i][i])

    # setw is for displaying nicely
    print("Lower Triangular\t\tUpper Triangular")

    # Displaying the result :
    for i in range(n):

        # Lower
        for j in range(n):
            print(lower[i][j], end="\t")
        print("", end="\t")

        # Upper
        for j in range(n):
            print(upper[i][j], end="\t")
```

```
        print("")


# Driver code
mat = [[2, -1, -2],
    [-4, 6, 3],
    [-4, -2, 8]]

luDecomposition(mat, 3)

# This code is contributed by mits

```

```
Lower Triangular                Upper Triangular
1       0       0               2       -1      -2
-2      1       0               0       4       -1
-2      -1      1               0       0       3
```

In [11]:

```python
# Python3 Program to decompose
# a matrix into lower and
# upper triangular matrix
MAX = 100


def luDecomposition(mat, n):

    lower = [[0 for x in range(n)]
             for y in range(n)]
    upper = [[0 for x in range(n)]
             for y in range(n)]

    # Decomposing matrix into Upper
    # and Lower triangular matrix
    for i in range(n):

        # Upper Triangular
        for k in range(i, n):

            # Summation of L(i, j) * U(j, k)
            sum = 0
            for j in range(i):
                sum += (lower[i][j] * upper[j][k])

            # Evaluating U(i, k)
            upper[i][k] = mat[i][k] - sum

        # Lower Triangular
        for k in range(i, n):
            if (i == k):
                lower[i][i] = 1 # Diagonal as 1
            else:

                # Summation of L(k, j) * U(j, i)
                sum = 0
                for j in range(i):
                    sum += (lower[k][j] * upper[j][i])

                # Evaluating L(k, i)
                lower[k][i] = int((mat[k][i] - sum) /
                                  upper[i][i])

    # setw is for displaying nicely
    print("Lower Triangular\t\tUpper Triangular")

    # Displaying the result :
    for i in range(n):

        # Lower
        for j in range(n):
            print(lower[i][j], end="\t")
        print("", end="\t")

        # Upper
        for j in range(n):
            print(upper[i][j], end="\t")
```

```
        print("")


# Driver code
mat = [[2, -9, 8],
    [-1, 5, 3],
    [1, 2, 3]]

luDecomposition(mat, 3)

# This code is contributed by mits

```

```
Lower Triangular                Upper Triangular
1       0       0               2       -9      8
0       1       0               0       5       3
0       0       1               0       0       3
```